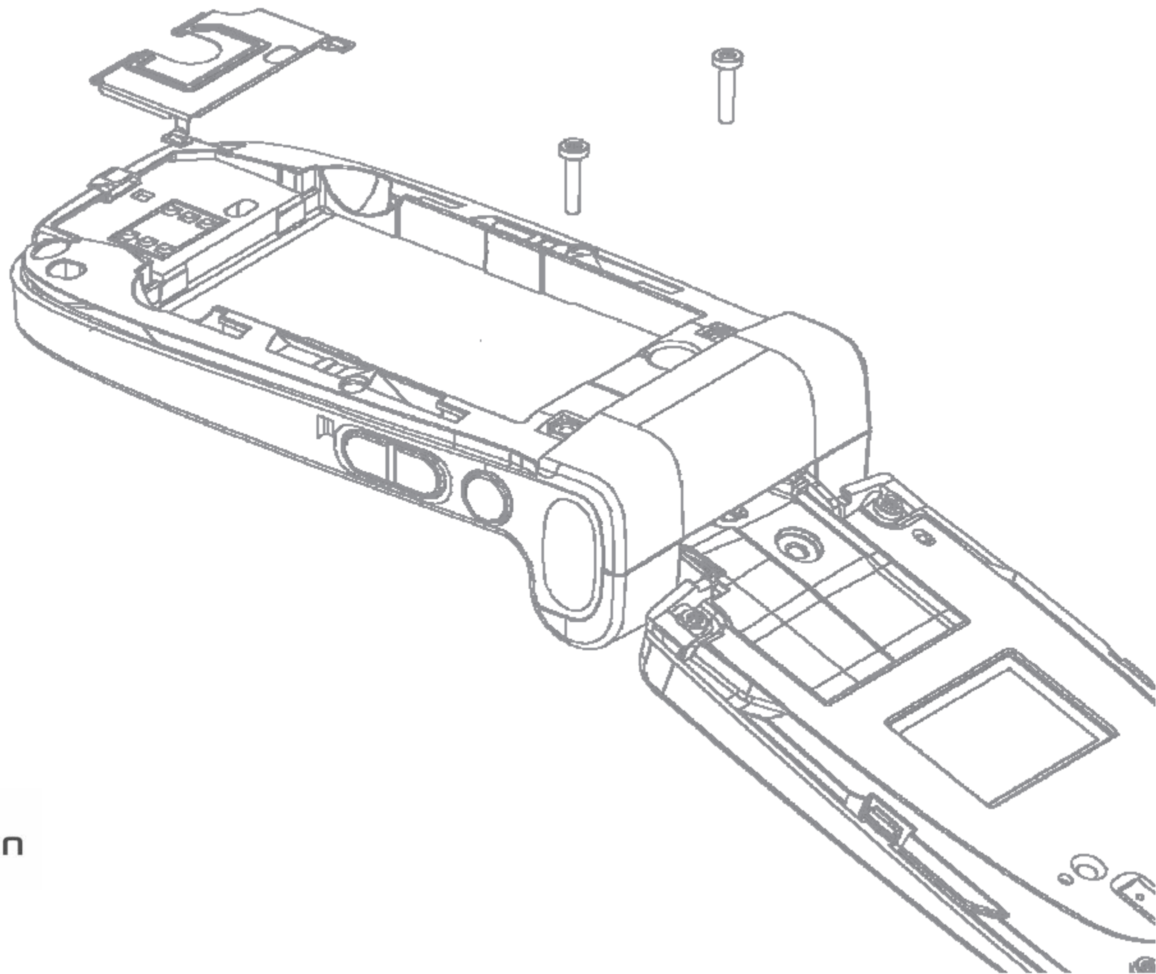
专业课题文章

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

2008年10月

索尼爱立信 Mobile JUnit 的单元测试

针对 Java Platform™, Micro Edition



Sony Ericsson

前言

本文档的用途

这篇专业课题文章介绍了 Mobile JUnit，它是一种针对 Java™ ME Platform 的单元测试框架；适用于索尼爱立信的智能手机和信息传递设备。无需具备单元测试经验，但是必须了解 Java ME 开发过程以便从本文档中受益。

本指南分为以下各章：

- 简介解释什么是单元测试，以及为什么单元测试是 Java ME Platform 上一项具有挑战性的任务。
- 利用 **JUnit** 进行单元测试概述 JUnit 单元测试框架，它是由索尼爱立信开发的 Mobile JUnit 框架的基础。
- **Mobile** JUnit 介绍详细介绍什么是 Mobile JUnit，它与标准 JUnit 框架有何不同，以及它如何增强标准 JUnit 框架。
- 编写单元测试说明如何使用 Mobile JUnit 创建单元测试。
- 集成测试讨论如何使用 Ant、 Eclipse 和 NetBeans 等常用工具将 Mobile JUnit 集成到构建过程中。
- 机上测试说明如何在索尼爱立信手机上运行单元测试。

阅读完本指南后，您将能够使用 Mobile JUnit 框架对 Java ME 应用程序进行单元测试。

该专业课题文章的出版商为：

Sony Ericsson Mobile Communications AB,
SE-221 88 Lund, Sweden

电话：+46 46 19 40 00 传真：+46 46 19 41 00
www.sonyericsson.com/cn

© Sony Ericsson Mobile Communications AB，2006。保留所有权利。在此授予您下载和 / 或打印本文档副本的许可。 任何未在此明确授予的权利都予以保留。

第一修订版 （2008年10月） 出版号：EN/LZT 108 8987 R1B

本文档由 Sony Ericsson Mobile Communications AB（索尼爱立信）出版，不提供任何担保 *。Sony Ericsson Mobile Communications AB （索尼爱立信）随时可能对本文档中的印刷错误、当前的不准确信息进行必要的改进和更改，或对程序和 / 或设 备进行改进，恕不另行通知。但是，这些更改将编 入本文档的新版本中。印刷版本仅视作临时的参考 副本。

* 所有暗示的担保，包括但不限于对适销性或针对特定用途的适用性的暗示担保，都不包含在内。对任何性质的意外或间接损坏，包括但不限于使用本文档中的信息而引起的利润或收入损失，索尼爱立信或其许可方概不负责。

索尼爱立信开发者世界

在www.sonyericsson.com/developer/china上，开发人员可以找到最新的技术文档和开发工具，如手机白皮书、针对不同技术领域，入门指南、 SDK （软件开发工具包）和工具插件的开发人员手册。 网站还包含新闻文章，提供免费技术支持的受管理论坛和一个共享专业知识和示例代码的Wiki社区。

有关这些专业服务的更多信息，请访问索尼爱立信开发者世界网站。

文档约定

印刷约定

代码以 Courier 字体编写：

```
public  class  MyTest  extends
     TestCase  { String  hello  =
     "hello";
...
```

文档历史记录

更改历史记录

2006-09-13	版本 R1A	文档在开发者世界上发布
2008-10-25	版本 R1B	新的文档布局。

目录

简介 5
单元测试 6
JUnit 测试框架 6
单元测试和 Java ME 7
利用 JUnit 进行单元测试 8
安装 8
测试用例 9
断言 11
固件 (Fixture) 12
测试套件 13
测试运行器 14
JUnit 和 Java ME 14
Mobile JUnit 介绍 15
安装 15
开始 15
Mobile JUnit 的工作方式 18
编写单元测试 19
单元测试的组织结构 19
测试用例 19
测试固件 20
测试套件 21
集成测试 22
Ant 22
IDE 集成 23
Eclipse 集成 23
NetBeans 集成 24
机上测试 25
准备手机 25
测试 26
结束语 27

简介

几乎没有开发人员愿意测试自己编写的代码 - 他们宁愿将测试工作委托给其他人，以便腾出时间来编写更多 代码。但是只有开发人员才最了解自己的代码，特别是代码的要求和限制。不熟悉代码的人会容易错过对开 发人员来说很明显的测试用例。定期测试自己代码的开发人员可以迅速发现和纠正错误，从而避免项目的其 余部分出现问题。

鼓励开发人员测试自己代码的最好方法是，进行可提供即时反馈的简单测试。单元测试是一种可以实现此目的的方法。

单元测试

单元测试 是一种程序，用于测试代码单元 的功能。通常情况下，单元是指较大程序中的单个过程、函数或方法（取决于正在使用的编程语言）。

单元测试必须非常全面，以测试代码行为的所有方面。对前提条件（调用代码之前 需满足的条件）和后置条件（调用代码之后 需满足的条件）进行测试，以确保正确的输入能够产生正确的结果，而意外的输入则导致相应的错误或异常。发现错误时，应根据情况调整相应的单元测试，以确保错误不会再次出现— 这称为回归测试。

单元测试的基本前提是整体正好等于各部分的集合：通过单元测试发现的错误不会传播到整个应用程序中，因为在应用程序中更难发现这些错误。并非应用程序行为的所有方面都可以通过单元测试进行检查，但是一组全面的单元测试可以确保应用程序的各个组件运行正常。开发人员能够将调试工作集中在组件之间的交互而不是组件本身上。

单元测试的弊端是它需要投入大量工作，而这些工作只能由所测试代码的开发人员完成。从头开始创建测试系统很重要，对图形应用程序而言尤其如此，并且对单个单元测试进行编码可能是一个非常漫长的过程。如果没有全面和易于使用的测试系统，开发人员很可能会完全跳过单元测试，而只依赖于应用程序测试来查找和纠正错误。

JUnit 测试框架

用 Java 编写的应用程序的单元测试主要通过 JUnit 框架完成。JUnit 是众多单元测试系统的一种，这些系统被统称为 *xUnit*，它们基于 Kent Beck 的想法，他针对 Smalltalk 编程语言编写了 SUnit 单元测试框架。

JUnit 是一个源代码开放的项目，自最初由 Erich Gamma 和 Kent Beck 实现以来已得到不断发展。JUnit 提供了许多可大大简化基于 Java 的单元测试的功能，其中包括：

- 断言 (Assertion)，以在测试之前或之后测试预期值。（这些断言功能与 Java 1.4 中引入的断言工具不同。）
- 测试固件 (fixture)，以模拟正在接受测试的代码正常运行所需的环境（包括所有必需的对象）。
- 测试套件 (suite)，以将各个测试用例组合到一起。
- 测试运行器 (runner)，以运行测试并捕获和报告测试是成功还是失败。

大多数情况下，创建一组单元测试也就是创建从某个 JUnit 测试用例类继承的类，并向该类添加新方法以执行各个单元测试。然后使用反射机制来运行测试。（Java 1.5 中引入的注解 (Annotation) 功能也可用来定义从 JUnit 4.0 版开始的单元测试的各个方面，但该功能在 Mobile JUnit 框架中不受支持。）

单元测试和 Java ME

测试 Java ME CLDC 环境中的代码存在许多困难。

在 CLDC 环境中，缺少诸如反射之类的基本语言功能，或者这些功能受到严重限制，因而很难开发测试框架。甚至是为使用 Connected Device Configuration (CDC) 的系统设计的代码也很难测试，因为这些代码使用了特定于平台的核心 Java ME 环境扩展。

最大的难题是运行要测试的代码。Java ME 应用程序必须在模拟器或适当的设备中运行。测试框架也必须在这些环境内运行，这非常有用 — 对于机上测试而言这特别重要，因为模拟器无法准确再现最终运行环境 的所有方面。

索尼爱立信的 Mobile JUnit 单元测试框架扩展了 JUnit 的功能而没有修改其代码或行为，从而实现了 Java ME CLDC 平台上的单元测试。与 JUnit 不同，它还包括基本的代码覆盖工具以帮助优化代码单元。

利用 JUnit 进行单元测试

要使用 Mobile JUnit，必须了解 JUnit 的工作方式以及单元测试的创建和运行方式。本章简要介绍基于 JUnit 的单元测试。如果已经熟悉 JUnit，请直接阅读下一章（第 15 页的“Mobile JUnit 介绍”）。

Mobile JUnit 基于 **JUnit 3.8.1**。JUnit 4.0 中引入了创建单元测试的新方法，这些方法目前与 Java ME 不兼容，因为它们依赖于 Java 1.5 中引入的新语言功能。本指南中的信息仅适用于 JUnit 3.8.1 版。

安装

JUnit 是源代码开放的 Java 软件。JUnit 的最新版本始终可以从 JUnit 主网站 http://www.junit.org 下载，但请注意， Mobile JUnit 已包括 JUnit 的适当版本 (3.8.1) — 完整安装说明将在下一章中介绍。

在 Java SE 平台上，安装过程就是解压缩 JUnit 分发包并将 junit.jar 文件置入 classpath 中。

测试用例

基本测试单元称为测试用例。测试用例是扩展 junit.framework.TestCase 的 Java 类，其中包括一种或多种测试方法。通常情况下，测试用例中的所有测试都是相关的；不相关的测试应放入单独的测试用例中。按照约定，测试用例的类名称以 *Test* 或 *TestCase* 结尾，但这不是必需的。

测试方法本身必须是公共的，并且必须以 *test* 开头，以便能够通过反射自动发现它们。下面是一个包含两个空测试的简单测试用例：

```
package  mypackage;
import  junit.framework.*;

public  class  MyTest  extends
     TestCase  { String  hello  =
     "hello";
     String  goodbye  =  "goodbye";

     public  void  testHello(){
     }

     public  void  testGoodbye(){
     }
}
```

当然，通常情况下，测试方法不为空，但是使用断言 可测试特定操作，例如：两个对象是否相等以及预期的布尔条件是否为 true。下面是测试用例的完整版本：

```
package  mypackage;
import  junit.framework.*;

public  class  MyTest  extends
     TestCase  { String  hello  =
     "hello";
     String  goodbye  =  "goodbye";

     public  void  testHello(){
          assertEquals(  "hello",  hello  );  //  OK
          assertEquals(  "hello",  goodbye  );  //  fails
     }

     public  void  testGoodbye(){
          assertEquals(  "goodbye",  goodbye  );  //  OK
          assertEquals(  "goodbye",  hello  );  //  fails
     }
}
```

运行测试后（将在后面讨论），其结果如下所示：

```
.F.F
Time:  0.01
There  were  2  failures:
1)  testHello(MyTest)junit.framework.ComparisonFailure:  expected:<hello>
but was:<goodbye>
     at  MyTest.testHello(MyTest.java:9)
     at  sun.reflect.NativeMethodAccessorImpl.invoke0(Native  Method)
     at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
     at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.j
ava:25)
2)  testGoodbye(MyTest)junit.framework.ComparisonFailure:  expected:<goodbye>
but  was:<hello>
     at  MyTest.testGoodbye(MyTest.java:14)
     at  sun.reflect.NativeMethodAccessorImpl.invoke0(Native  Method)
     at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
     at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.j
ava:25)

FAILURES!!!
Tests  run:  2,              Failures:  2,     Errors:  0
```

这两种失败情况（包括预期值和实际值）会在报告中明确说明。

断言

JUnit 定义了多个断言以简化测试。这些断言与 Java assert 关键字无关 （它们是从 TestCase 的超类继承的方法），但是工作方式类似。每个断言都测试一个特定条件，如果条件不满足，则引发异常。框架将捕获这些异常，记录错误并继续进行下一个测试。

保持每种测试方法内的断言数目最少。否则，第一次失败将导致无法对测试方法的剩余部分进行测试。

可以断言 7 个条件：

- **assertEquals** 测试预期值和实际值是否相等。将重载此方法以支持每个基本 Java 类型以及对象引用， 使用 equals 方法测试后者。
- **assertFalse** 期望布尔值为 false。
- **assertNotNull** 期望对象引用不为 null。
- **assertNotSame** 期望两个对象引用不引用同一对象。 （这与测试不等性不同。两个对象可以相等也可以不相等，只关注它们是否是不同的 对象。）
- **assertNull** 期望对象引用为 null。
- **assertSame** 期望两个对象引用引用同一对象。 （同样，这与测试等同性不同。）
- **assertTrue** 期望布尔值为 true。

如果断言失败，则会引发 junit.framework.AssertionFailedError 或其子类 junit.framework.ComparisonFailure。由于这些异常是从 java.lang.Error 派生的，因此它们通常不由正常异常处理程序捕获。

框架会自动捕获此测试方法引发的任何异常，并将其按失败处理，因此可能发生的任何经检查的异常都可以照常在方法定义中声明。 （不必声明 JUnit 框架引发的异常，因为它们是未经检查的异常。）

如果没有合适的断言，则对条件进行测试并在预期条件不满足时使用 fail 方法。fail 方法会无条件引发 AssertionFailedError。如果未引发预期异常，则使用 fail 方法：

```
public  void  testNotInt(){
     String  value  =  ....  //  some  string  we  want
     to  test try  {
          int  v  =  Integer.parseInt(  value  );
          fail(  value  +  "  is  not  supposed  to  be  an  int"  );
     }
     catch(  NumberFormatException  e  ){
          //  do  nothing,  this  is  what  we  expect  to  happen
     }
}
```

每个断言方法（包括 fail 方法）都具有另外一种重载形式，该形式将一个字符串作为它的第一个参数。使用该字符串代替在引发 AssertionFailedError 或 ComparisonFailure 时出现的一般错误消息。

固件（Fixture）

由单元测试所测试的代码通常依赖于特定对象是否存在。测试固件可用来创建执行代码所需的环境。最简单的固件是静态的或者是测试用例类中定义的实例变量。但是对于更复杂的情形，测试用例可以改写 setUp 和 tearDown 方法，如下所示：

```
import junit.framework.*;
import java.io.*;

public class MyTest2 extends TestCase {
    private BufferedReader reader;

    public void setUp() throws Exception {
        reader = new BufferedReader( new FileReader( "data.txt" ) );
    }

    public void tearDown() throws Exception {
        try {
            reader.close();
        }
        catch( IOException e ){
        }

        reader = null;
    }

    // Run a test that loops through all the lines in the
    // data.txt file

    public void testReading1() throws
        IOException { String line;

        while( ( line = reader.readLine() ) != null ){
            // assert things here
        }
    }

    // Run a different test on the same data

    public void testReading2() throws
        IOException { String line;

        while( ( line = reader.readLine() ) != null ){
            // assert different things here
        }
    }
}
```

在运行每个测试之前，将调用 setUp 方法以准备测试环境。在测试运行之后，将调用 tearDown 方法以取消对测试环境的实例化。（请注意，即使由于异常或断言失败而导致测试失败，也会调用 tearDown。）这两个方法的默认实现机制都不执行任何操作。

测试套件

可以将测试用例组合到一个测试套件 中。测试套件是同时运行多个测试的简便方法。 与测试用例不同，测试套件不从基类继承，而是定义一个将返回 junit.framework.TestSuite 实例的静态方法 suite，该实例中已添加了各个测试用例：

```
import junit.framework.*;

public class MyTestSuite {
    public static Test suite(){
        TestSuite suite = new TestSuite();
        suite.addTestSuite( MyTest.class );
        suite.addTestSuite( MyTest2.class );
        return suite;
    }

    public static void main( String[] args ){
        junit.textui.TestRunner.run( suite() );
    }
}
```

注意：实际上声明 suite 方法是为了返回 junit.framework.Test。Test 实际上是由 TestCase 和TestSuite 实现的接口，这使得这两个类在大多数环境中可以互换。 测试套件可以嵌套，这样就可以根据类的包结构轻松地构建完整的测试层次结构。

按包来组织测试。虽然不要求在要测试的类所在的同一个包中创建测试用例和测试套件，但这么做非常有用。这允许您调用受保护和仅属于包的方法和数据。测试仍可以在单独的源目录中进行。

测试运行器

JUnit 框架的最后部分是测试运行器。测试运行器会运行测试并记录其输出。提供有可供控制台和图形界面（AWT 或 Swing）使用的测试运行器。要运行一组测试，只需要使用测试用例或测试套件的名称调用测试运行器使其运行即可：

```
java -cp .;c:\junit\junit.jar junit.textui.TestRunner MyTestSuite
```

`junit.textui.TestRunner` 类会将其结果输出到控制台。要以图形方式查看结果，请使用 `junit.awtui.TestRunner` 或 `junit.swingui.TestRunner`。

运行测试的简便方法是在每个测试类中包含一个主要方法。然后，该方法会调用运行器（通常是文本运行器），与前面部分的测试套件示例中的步骤相同。

JUnit 和 Java ME

如您所见，基于 CLDC 的 Java ME 环境中未提供 JUnit 所依赖的功能（如反射）。在这些平台上进行单元测试需要一个不同的测试框架，该框架由索尼爱立信的 Mobile JUnit 提供。

Mobile JUnit 介绍

安装

可以从 索尼爱立信开发者世界 网站的 Java 文档和工具部分下载 Mobile JUnit。Mobile JUnit 依赖于 Sun Java Wireless Toolkit for CLDC (WTK)，并且可与包含或扩展 WTK 的任何开发工具一起使用，如适用于 Java ME 平台的索尼爱立信 SDK。

安装索尼爱立信**SDK**。本文档假定您已经下载索尼爱立信 SDK（也可从开发者网站的 Java 文档和工具部分获得），并已将其安装到默认位置 C:\SonyEricsson\JavaME_SDK_CLDC。在较旧设备上开发 MIDP 1.0 时，建议使用 JDK 1.4.2，但对于较新设备和 / 或 MIDP 2.0 开发，则可以使用 JDK 1.5 或更高版本。只能在 Microsoft Windows 操作系统上使用 SDK。

要安装 Mobile JUnit，请运行下载的 ZIP 文件中的安装程序以调用安装向导。接受许可协议后，选中所有组件并进行安装。 （一定要选中 “生成脚本”选项，否则后面的说明将不起作用。还会提示您提供 Java SDK 的位置。）Mobile JUnit 将安装在 C:\SonyEricsson\JavaME_SDK_CLDC\Mobile_JUnit 文件夹中， 示例项目安装在 C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps 文件夹中 （示例 项目名为 mobile-ju-sampleproject）。本文档将这些文件夹简单地称为 “Mobile JUnit 文件夹”和“示例项目文件夹”。

开始

应在安装 Mobile JUnit 后立即使用它测试示例项目。示例项目是标准 Sun Java Wireless Toolkit (WTK) 项目，您可以在 src 文件夹中找到要测试的类 （CharacterCounter 类）的源代码：

```
public class CharacterCounter {

    private boolean ignoreCase = false;

    public CharacterCounter() {
        this(false);
    }

    public CharacterCounter(boolean ignoreCase) {
        this.ignoreCase = ignoreCase;
    }

    /**
    * Counts the number of occurences of a
    * character in a string.
```

```
    */
    public int count(char ch, String str) {

        if (ignoreCase) {
            ch = Character.toUpperCase(ch);
            str = str.toUpperCase();
        }

        int result = 0;
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == ch) {
                result++;
            }
        }

        return result;
    }

};
```

还可以在项目文件夹中找到一个测试文件夹，以及一个简单测试用例的源代码，该测试用例包含 CharacterCounter 类的两个测试：

```
import com.sonyericsson.junit.framework.TestCase;

public class SampleTest extends TestCase {

    public void testCountCharacter() {
        CharacterCounter counter = new
        CharacterCounter(); assertEquals(0,
        counter.count('a', "Hello")); assertEquals(1,
        counter.count('a', "a")); assertEquals(0,
        counter.count('a', "A")); assertEquals(2,
        counter.count('a', "Attackaz!"));
    }

    public void testCountCharacterIgnoreCase() throws
        Exception { CharacterCounter counter = new
        CharacterCounter(true); assertEquals(0,
        counter.count('a', "Hello")); assertEquals(1,
        counter.count('a', "a"));
        assertEquals(1, counter.count('a', "A"));
        assertEquals(3, counter.count('a', "Attackaz!"));
    }
}
```

请注意， Mobile JUnit 测试用例与标准 JUnit 测试用例几乎完全相同，唯一区别在于，前者扩展 com.sonyericsson.junit.framework.TestCase 而不是 junit.framework.TestCase。

要运行测试，请打开控制台窗口，并将当前目录设置为 Mobile JUnit 文件夹。可使用 run-mobile-junit 批处理文件，来编译和运行它在指定项目的测试文件夹中找到的任何测试用例。使用以下命令运行测试：

```
set projects=c:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps

run-mobile-junit --project-dir:%projects%\mobile-ju-sampleproject
    --device:SonyEricsson_W800_Emu --compile-midlet:yes
```

这会生成和编译基于 MIDlet 的测试运行器，启动索尼爱立信 W800 模拟器并运行测试。

> 也可以使用其他模拟器。提供了许多其他模拟器，可以在 C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\wtklib\devices 文件夹中找到完 整列表。请注意，要模仿的设备的名称必须与设备文件夹中以 "_Emu"（区分大小写）为后缀的某个文件夹名称（如上所示）相匹配。

模拟器在运行测试后会自动关闭并输出测试结果：

```
Building  midlet...  221ms
Building  test  midlet...  982ms
Uploading  and  running  test  midlet...  2s  434ms
..Running  with  storage  root  SonyEricsson_W800_Emu
Execution  completed.
0  bytecodes  executed
0  thread  switches
823  classes  in  the  system  (including  system  classes)
0  dynamic  objects  allocated  (0  bytes)
0  garbage  collections  (0  bytes  collected)Done.  10s  976ms

Time:  10.976

OK  (2  tests)
```

更改 CharacterCounter 类以返回不正确的值并再次运行测试：

```
Building  midlet...  210ms
Building  test  midlet...  961ms
Uploading  and  running  test  midlet...  2s  404ms
.F.FDone.  5s  468ms

Time:  5.468
There  were  2  failures:
1)  <untitled>/SampleTest/
testCountCharacter(SampleTest)junit.framework.AssertionFailedError:
expected:[1]  but  was:[2]
     at  SampleTest.testCountCharacter(:0)
     at  SampleTestRunner.runTest(:0)
     at  com.sonyericsson.junit.framework.SingleTest.run(:0)
     at  com.sonyericsson.junit.framework.ClassnameTest.run(:0)
     at  com.sonyericsson.junit.framework.JARTestSuite.run(:0)
     at  com.sonyericsson.junit.midletrunner.StdoutMidlet.run(:0)
2)  <untitled>/SampleTest/testCountCharacterIgnoreCase(SampleTest)Running
with storage  root  SonyEricsson_W800_Emu
Execution  completed.
0  bytecodes  executed
0  thread  switches
823  classes  in  the  system  (including  system  classes)
0  dynamic  objects  allocated  (0  bytes)
0  garbage  collections  (0  bytes
collected)junit.framework.AssertionFailedError:
expected:[1]  but  was:[2]
     at  SampleTest.testCountCharacterIgnoreCase(:0)
     at  SampleTestRunner.runTest(:0)
```

```
      at  com.sonyericsson.junit.framework.SingleTest.run(:0)
      at  com.sonyericsson.junit.framework.ClassnameTest.run(:0)
      at  com.sonyericsson.junit.framework.JARTestSuite.run(:0)
      at  com.sonyericsson.junit.midletrunner.StdoutMidlet.run(:0)

FAILURES!!!
Tests  run:  2,              Failures:  2,     Errors:  0
```

输出与使用 JUnit 测试失败时看到的结果非常类似。

Mobile JUnit 的工作方式

对于编写单元测试的开发人员而言， Mobile JUnit 和 JUnit 非常类似。在这两个框架中，测试用例和测试套件几乎是以完全相同的方式编写的，主要区别在于 Mobile JUnit 使用了 com.sonyericsson.junit.framework 包而非 junit.framework。但是，在测试的运行方式方面这 两者却有着重要区别。

JUnit 框架在运行时使用 Java 内置的反射功能来查找并运行测试用例内的所有单元测试。但是，基于 CLDC 的环境缺少反射功能，这意味着不可能在运行时发现方法。Mobile JUnit 则使用生成的 helper 类来列出和调用各个测试。还会生成测试用例的主列表。

为了运行测试，会生成一个测试 MIDlet。MIDlet 的主类是使用测试用例列表运行测试的 Mobile JUnit 测试运行器。（请注意，也支持测试套件，在这种情况下，测试套件会指定要运行的测试用例。）测试运行器、测试用例、helper 类和 Mobile JUnit 框架本身一起打包到单个 JAR 文件中。还会生成 JAD 文件。设备模拟器会加载生成的 MIDlet 套件并启动 MIDlet 以运行测试。

Mobile JUnit 依赖于移动信息设备描述 （**Mobile Information Device Profile**，**MIDP**）。即使测试本身不依赖于 MIDP，仍然假定 MIDP 环境中会发生运行时测试。

Mobile JUnit 也支持使用索尼爱立信手机进行机上测试。无需下载测试并将其安装到设备上，因为专门的代理应用程序会为您执行此操作。机上测试将在下文进行介绍。

编写单元测试

由于 Mobile JUnit 框架与 JUnit 框架几乎完全相同，因此很容易创建单元测试。

单元测试的组织结构

单元测试源代码默认位于项目目录的 test\src 文件夹中。项目目录通常位于 Wireless Toolkit apps 文件夹中，例如：

c:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\wtk2\apps\MyProject

应用程序的源代码通常位于 src 文件夹中，默认情况下，Mobile JUnit 会在该文件夹中查找源代码。单元测试源代码应按该文件夹内的包进行组织，这与应用程序源代码的组织方式类似。

编译测试用例时， Mobile JUnit 会创建 test\bin 文件夹以存放所有生成的文件。这可以确保单元测试不会妨碍正在测试的应用程序或库的正常编译和运行。

如果默认文件夹位置不合适，可使用命令行选项指定新位置。要查看命令行选项的完整列表，请参考索尼爱立信的 Java ME CLDC 开发人员指南 （可从索尼爱立信开发者世界获得）。

测试用例

如上所述，Mobile JUnit 测试用例与 JUnit 测试用例几乎完全相同。如果不需要固件，测试用例仅导入不同的包：

```
package  mypackage;
import  com.sonyericsson.junit.framework.*;

public  class  MyTest  extends  TestCase  {
    public  void  testMyStuff(){
       ....  //  tests  go  here
    }
}
```

Mobile JUnit 假定任何名称以 *test* 开头且不带参数的方法都是测试方法。

由 JUnit 定义的所有 assert 和 fail 方法均可用于 Mobile JUnit 测试用例。还定义了 AssertionFailedError 和 ComparisonFailure 异常。JUnit 和 Mobile JUnit 测试用例之间的唯一实 质性区别在于，测试固件实例化和取消实例化的方式。

测试固件

Mobile JUnit 在其 TestCase 类的实现机制中定义了 setUp 和 tearDown 方法。在需要对测试固件进行实例化或取消实例化时，可以使用这些方法。这些方法与 JUnit 定义的方法几乎完全相同，只是定义它们的目的是引发 Throwable 而不是 Exception。与在 JUnit 中一样，在测试之前立即调用 setUp，并在测试之后立即调用 tearDown。

> 清理测试固件。请记住，不支持在基于 CLDC 的环境中终结对象。应在 tearDown 方法中明确释放或关闭 setUp 方法中所使用或打开的任何资源。

需要 MIDlet 主类的测试可以使用 setUp 的重载版本，以访问运行测试用例的 javax.microedition.midlet.MIDlet 实例：

```
package mypackage;
import com.sonyericsson.junit.framework.*;
import javax.microedition.midlet.*;

public class MyTest extends TestCase {
    private MIDlet _midlet; // store for test case use

    public void setUp( MIDlet midlet ) throws Throwable {
        _midlet = midlet;
    }

    public void testMIDletNotNull(){
        assertNotNull( _midlet );
    }

    public void tearDown() throws Throwable {
        _midlet = null;
    }
}
```

注意：您只能重写 setUp 方法的一种形式。

测试套件

与测试用例一样， Mobile JUnit 测试套件与 JUnit 测试套件也几乎完全相同：

```
package mypackage;
import com.sonyericsson.junit.framework.*;

public class MyTestSuite extends TestSuite {
    public MyTestSuite(){
        addTestSuite( MyFirstTest.class );
        addTestSuite( MySecondTest.class );
    }
}
```

就像在 JUnit 中一样，测试套件可以调用其他测试套件以及单个测试用例：传递给 addTestSuite 的类只需要实现 Test 方法，而 TestCase 和 TestSuite 都要实现该方法。

但是，与 JUnit 不同的是，通常不在测试套件中包含 main 方法以自动调用测试运行器。而是通过在 Mobile JUnit 命令行上使用 --suite 选项来运行测试套件：

```
run-mobile-junit --project-dir:<your SDK project path>\mobile-ju-
sampleproject
     --device:SonyEricsson_W800_Emu --compile-midlet:yes
     --suite:MyTestSuite
```

--suite 选项也可以用于运行单个测试用例，方法是指定测试类（而非测试套件）的完全限定名称。 如果不指定任何套件， Mobile JUnit 将运行它在测试源文件夹中找到的所有测试用例。

集成测试

虽然总是可以使用 run-mobile-junit 从命令行运行测试，但更好的方法是将单元测试集成到应用程序的构建过程中。Java ME 开发人员通常使用的三种构建工具是：Ant、 Eclipse 和 NetBeans。

Ant

使用 Ant Java 任务可以轻松运行单元测试。首先，定义一个引用 Mobile JUnit 和 JUnit jar 文件的 path 元素：

```
<path  id="test-classpath"  >
     <pathelement  path="${junit-jar}"  />
     <pathelement  path="${mobile-junit-jar}"  />
</path>
```

然后定义 Mobile JUnit 的目标：

```
<target  name="run-javame-tests"  >
     <java
          classname="com.sonyericsson.sdkme.junit.OnDeviceTest"
          fork="true"  failonerror="true"  >
          <classpath  refid="test-classpath"  />
          <arg  value="--javac:${javac}"  />
          <arg  value="--project-dir:${project-dir}"  />
          <arg  value="--device:SonyEricsson_W800_Emu"/>
          <arg  value="--compile:true"  />
          <arg  value="--compile-midlet:true"  />
     </java>
</target>
```

要运行测试，只需在 Ant 命令行上指定 run-javame-tests 目标即可：

```
ant  run-javame-tests
```

当然，这假定上面显示的构造段中的 javac 属性引用的是有效 Java SDK，并且 project-dir 设置为项目目录。

IDE 集成

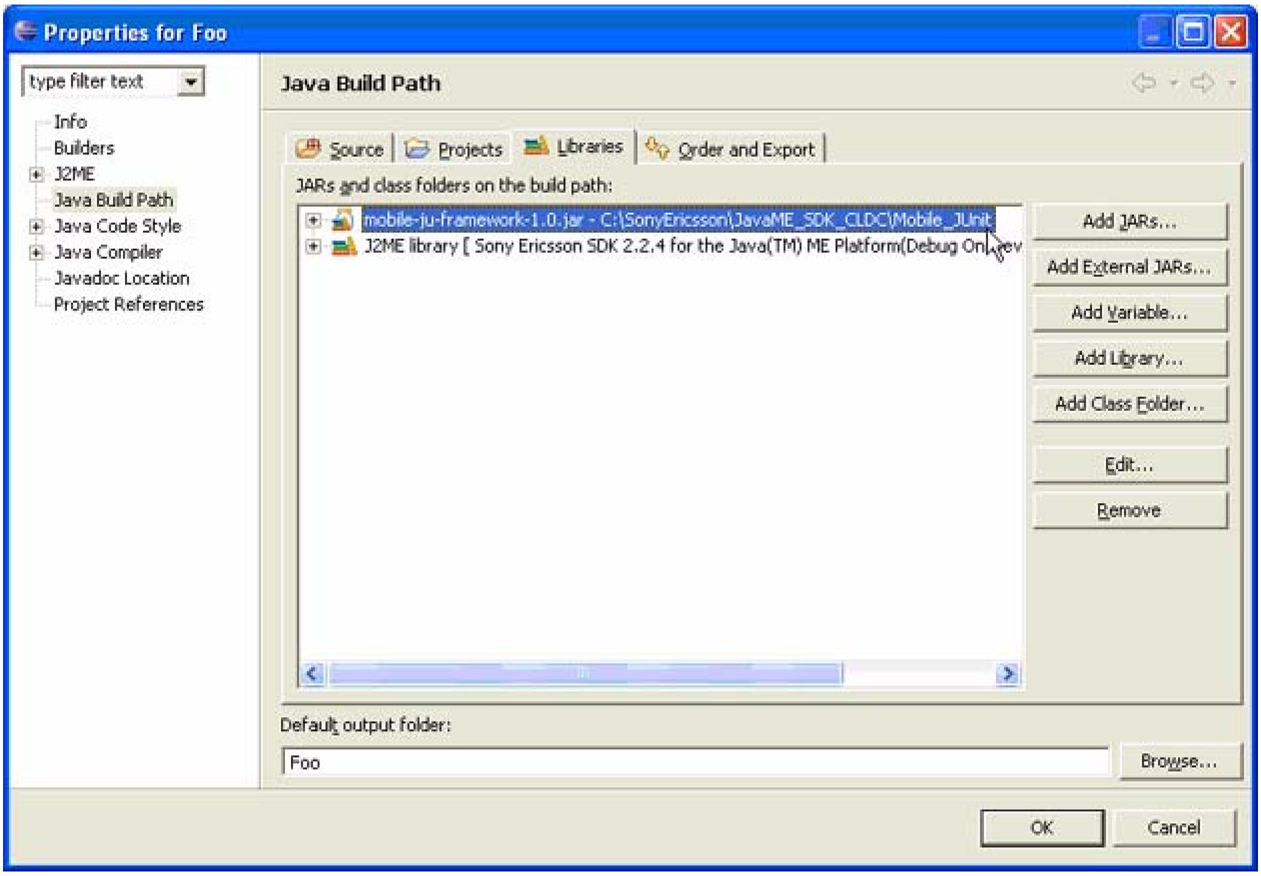
许多开发人员更喜欢使用 Eclipse 或 NetBeans 之类的集成开发环境 (IDE) 来开发 Java ME。在这些环境中使用 Mobile JUnit 非常简单。

Eclipse （安装了 EclipseME 插件）和 NetBeans 都使用 Ant 构建项目。只需将适当的 Mobile JUnit 目标添加到项目构建文件中即可，如前一部分所示。但是，这不允许您使用 IDE 创建或编辑测试用例，因为项目路径中缺少必需的类文件。要将 Mobile JUnit 与 IDE 正确集成，必须将 mobile-ju-framework-1.0.jar 添加到项目 classpath 中。可以在索尼爱立信 SDK 的 Mobile_JUnit 文件夹中找到该 jar 文件。

> 仅添加框架。无需将在同一文件夹中找到的 mobile-ju-1.0.jar 和 junit.jar 添加到 IDE classpath 中。这些文件将由 Mobile JUnit 独占使用以编译和运行测试，并且包含对非 ME 类的多个引用。

Eclipse 集成

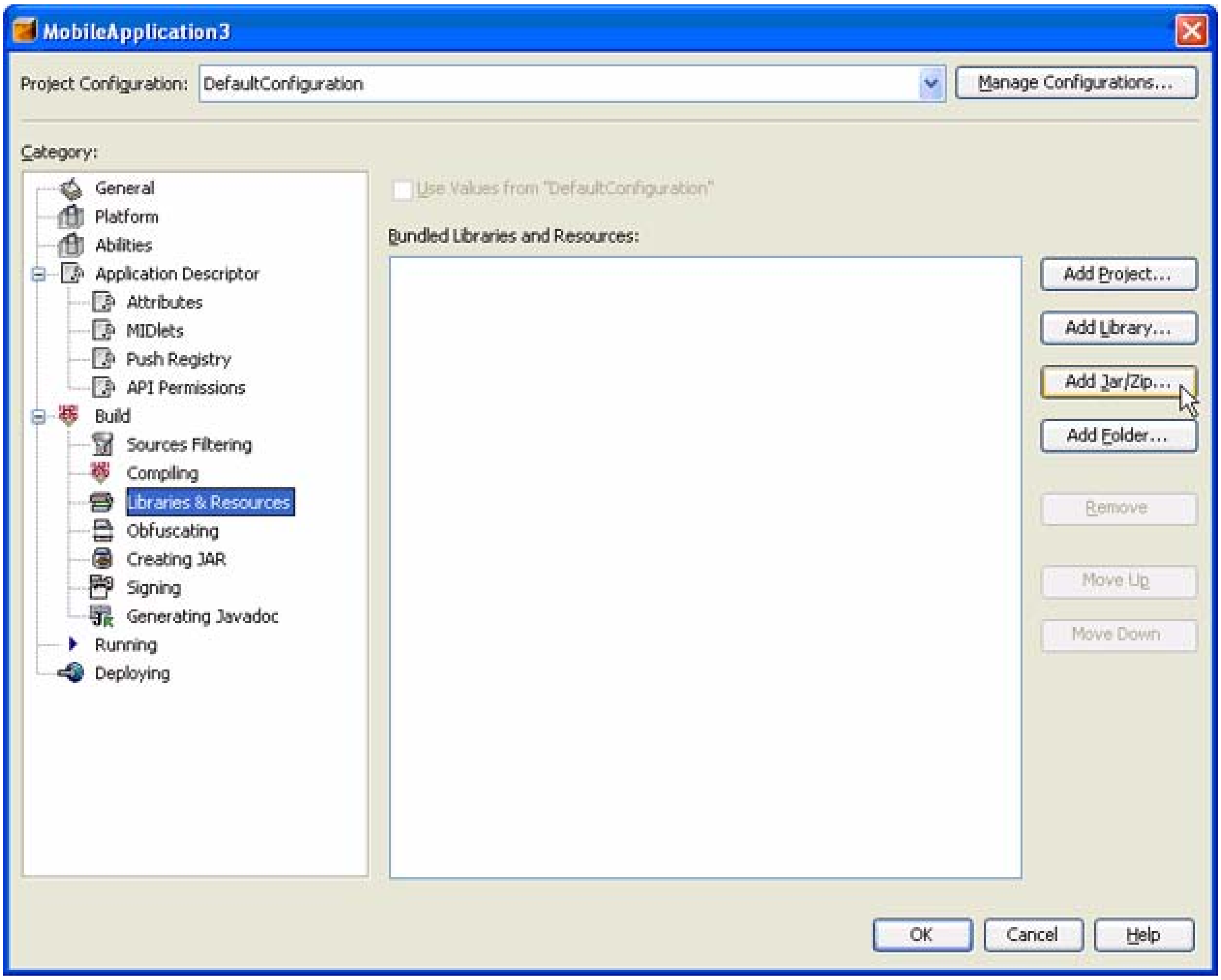
Eclipse 用户必须将框架 jar 文件添加到项目 classpath 中。在项目设置对话框中，导航到 **Java Build Path**（Java 构建路径），选择 **Libraries** （库）选项卡，然后按 **Add External Jars** （添加外部 Jar）按钮。找 到框架 JAR 文件并将其添加到构建路径中：



现在就可以在 Eclipse 环境中创建和编辑单元测试了。

NetBeans 集成

向 NetBeans 中添加框架与上面的方法类似。打开项目属性对话框并导航到 **Build** （构建）下的 **Libraries & Resources** （库和资源）。按 **Add Jar/Zip** （添加 Jar/Zip）按钮。找到框架 JAR 文件并将其添加到构建路径中：



现在就可以在 NetBeans 环境中编辑单元测试了。

机上测试

索尼爱立信 Mobile JUnit 的在机单元测试与所有索尼爱立信手机中的在机调试功能使用相同的机制，并且可以对在机调试功能起补充作用，同时它还不局限于仿真设备。测试是完全自动完成的 — 索尼爱立信 Mobile JUnit 将自动在设备上安装和运行测试，并捕获生成的输出以供分析。

机上测试仅限于索尼爱立信手机。尽管 Mobile JUnit 可以在任何与 Sun Java Wireless Toolkit 兼容的设备模拟器上使用，但只能与用于 Java ME 平台的索尼爱立信 SDK 一起在索尼爱立信手机上执行机上测试。

注意：使用 K610 中引入的 Java Platform 7，可以分别使用通用 Java Platform 7 模拟器和调试配置文件处理在机调试和仿真。有关手机与 Java 平台的关系的详细信息，请参考索尼爱立信 Java ME CLDC 开发人员指南 （可从索尼爱立信开发者世界获得）。

准备手机

尝试执行机上测试之前，确保您的手机型号受支持。可在以下位置找到受支持设备的列表：

c:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug\Devices 在测试开始之前，通过串行线缆将

手机连接到计算机并激活索尼爱立信串行代理：

c:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug\bin\serialproxy.exe 串行代理应用程序会将

索尼爱立信 Mobile JUnit 链接到物理设备。 注意：必须为机上测试选择适合实际设备的在机调试配

置文件，这一点与在模拟器上的测试不同。

测试

连接手机并运行代理后，按通常方式运行测试用例（或测试套件）。唯一不同之处在于，要通过设置 --wtk 选项来使用设备上的 Wireless Toolkit 版本：

```
run-mobile-junit  --project-dir:%projects%\mobile-ju-sampleproject
    --device:SonyEricsson_JP-7  --compile-midlet:yes
    --wtk:C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug
```

编译完成后，加载测试并在连接的设备上运行测试。

结束语

单元测试是用于查找错误并防止错误重新潜入应用程序代码的重要工具。索尼爱立信 Mobile JUnit 通过针对 CLDC 环境的限制调整 JUnit，从而使得 Java ME 开发人员能够轻松地进行单元测试。请立即向项目中添加单元测试，并且一定要查看索尼爱立信开发者世界，以获取 Mobile JUnit 的更多更新。